Generic Systolic Array for Run-time Scalable Cores

Andrés Otero, Yana E.Krasteva, Eduardo de la Torre and Teresa Riesgo

Centro de Electrónica Industrial. Universidad Politécnica de Madrid
{andres.otero, yana.ekrasteva, eduardo.delatorre, teresa.riesgo}@upm.es

Abstract. This paper presents a scalable core architecture based on a generic systolic array. The size of this kind of cores can be adapted in real-time to cover changing application requirements or to the available area in a reconfigurable device. In this paper, the process of scaling the core is performed by the replication of a single processing element using run-time partial reconfiguration. Furthermore, rather than restricting the proposed solution to a given application, it is based on a generic systolic architecture which is adapted using a design flow which is also proposed. The paper includes a related work discussion, the proposal and definition of a systolic array communication approach, which does not require the use of specific macro structures and permits to achieve higher flexibility, and a design flow used to adapt the generic architecture. Further, the paper also includes an image filter application as a simple use case, along with implementation results for Virtex 5 FPGA.

Keywords: Digital signal processing, adaptable cores, scalability, systolic array, partial runtime reconfiguration.

1 Introduction

Current multimedia applications are offered on heterogeneous terminals, with a broad range of features, using different communication networks and variable bandwidth availability capabilities [1]. As a result, single devices are supposed to deal with multiple coding standards, which evolve and emerge in short time. Consequently, devices lifetime is shortened and their replacement with new ones, with advanced features, requires speeding up time-to-market.

This challenge can be solved providing more flexibility to devices by including adaptability capabilities. Device adaptation can be based on different parameters, like the battery level, the available computational power, the target coding standard or even a profile within a standard. The need of adaptability could be easily fulfilled by the use of software implementations. However, most of the multimedia related tasks are compute-intensive and demand high performance and fast execution, which can be achieved in hardware. In this context, reconfigurable computing can fulfill both, performance and flexibility, requirements.

Among the flexibility requirements, there is a wide interest in proving solutions that permit to scale in real-time the functionality of a hardware block. Functional scaling is achieved by modifying the size of the operation performed by a core, depending on the application requirements at a given moment. Such solutions can be advantageous in

many domains. Among others, in coding standards, where scaling is oriented to variable-size hardware operations, like the Discrete Wavelet Transform (DWT) presented in [2], the variable-size Discrete Cosine Transform (DCT) in [3] or in motion estimation and filters [4], and also, in tasks scaling for multi-standard communication systems [5] and [6].

This paper addresses a solution where the functional scalability of a hardware block is achieved by means of spatially scaling the physical implementation of a core. This means modifying the area occupied by the core inside a reconfigurable system. In addition, with this kind of solutions, different tradeoffs between the area occupied by a core and its performance can be set. An example of such system can be found in the scalable window based image filter proposed in [7], or in the scalable DCT presented in [8].

A direct approach to create variable-size scaling cores is to implement the same task in several cores, with different performance and area requirements, and load a suitable one in the system depending on the available hardware resources. Differently, highly parallel, modular and regular architectures have been studied as a scalable core architecture alternative to reduce the overhead of the adapting process. These architectures can be scaled by means of the addition and removal of parallel blocks resulting in lower adaptation times. Among the architectures with these characteristics, scalable cores based on systolic arrays and distributed arithmetic are the most common, like the ones presented in [9] and [10]. Distributed arithmetic provides scalable solutions to perform arithmetic operations, while systolic architectures can solve full computing-intensive tasks in a broad range of fields. An interesting summary of different systolic arrays, each one for a specific application field, can be found in [11].

There are several alternatives to implement the process of scaling a core, like the use of parameterizable HDL code, which results in different core implementations once synthesized [12], or to use a clock gating technique for the unused elements [3]. However, the first solution does not permit real-time adaptation, while the second one does not release the unused logic.). Therefore, the exploitation of partial run-time reconfiguration capabilities of state of the art Field Programmable Gate Arrays (FPGA) is the widely adopted solution, since it overcomes these limitations.

This paper focuses on systolic-array-based scalable cores that permit run-time adaptability. However, differently from the related work discussed in the paper, it presents a general systolic architecture that can be customized, using a proprietary design flow, to solve concrete problems. The proposed solution permits, using a single processing element replication process, to scale the functionality of a core mapped at run-time, or even to create a new one. The replication of the basic element is carried out by means of dynamic reconfiguration. Additionally, an approach to communicate single reconfigurable elements of the array, which does not require the use of specific macro structures, is provided. The proposed communication approach permits to provide generality to the systolic array, to gain flexibility and also reduces the run-time reconfigurable systems implementation area overhead.

The rest of the paper is organized as follows. In section 2, the related work is described, highlighting the main differences with the proposed solution, which is presented in detail in section 3. Section 4, provides implementation results and a use case of the proposed architecture and finally, conclusions can be found in section 5.

2 Related Work

In this section, a review of run-time scalable cores based on systolic arrays is included. Some representative related works in this specific topic have been selected and characterized based on two main criterions. The first criterion is related to the array implementation and its floorplaning on the dynamically reconfigurable system, which defines the overall system flexibility. In this aspect, one and two-dimensional solutions can be differentiated. The second criterion, which is related to the system generality and also influence in its flexibility, is the partial dynamic reconfiguration design flow used.

The scalable FIR filter introduced in [13] is an example of a one-dimensional architecture, which is scaled by means of the addition of independent modules. Each additional module increases the number of coefficients of the filter, adapting the filter response to the desired filtering mask in real-time. This is a good example of tradeoff between the filter response quality and resource occupation. In this work, the design flow was the Xilinx Modular Design, which restricted system flexibility significantly.

In contrast, the newer Early Access Partial Reconfiguration (EAPR) flow, also provided by Xilinx, has been selected in [14] and [8]. The first work proposes a Scalable two-dimensional DCT architecture, where the EAPR is used to: i) adapt the precision of the DCT coefficients and ii) to remove and/or add processing elements to achieve different types of zonal coding, from 1×1 up to 8×8. That work also allows the use of the area of the removed elements by other tasks, but in a restricted manner. The second work, that is also two-dimensional, introduces an interesting reconfigurable DCT where processing elements are modified in the FPGA at run-time according to the zigzag order. In that work, elements are added or removed depending on the desired compression quality.

Two main conclusions can be drawn from the analysis of the related work. First, all the proposals are focused on offering solutions to specific problems or applications. Apart from the ones described in this section, other examples of concrete purpose scalable works are the template matching reconfigurable architecture presented in [15], the FIR filter in [4] or the image filter in [7]. Second, the reconfigurable architecture implementations and the design flows do not permit to completely release the area that is not used by the core. Therefore, the use of this unused area is highly restricted and permits to load only a narrow set of cores or processing elements. On the contrary, this paper provides a general approach to create any scalable, two-dimensional and run-time reconfigurable systolic array architecture. Furthermore, the solution permits to load a broad type of processing elements and cores in the system, cores, which might belong to different applications.

The limitations that derive from the selected design flow, Modular Design or EAPR, will be further discussed in section 3 and section 4.

3 Scalable Systolic Array

Systolic arrays can be defined as pipelined arrays of processing elements that rhythmically compute and pass data through its structure. Differently from the related work, the approach adopted in this paper is the definition of a generic systolic array that is customized afterwards following a design flow described in subsection 3.2. The basic idea behind the generic systolic array is the definition of a unique fine or medium grain

processing element that is replicated in two dimensions. This process is used for building new systolic arrays based scalable cores or for scaling up/down existing ones. This results not only in the scaling of the functionality and/or the area of the core, but also, due to the implementation solution proposed in subsection 3.1, permits to free the remaining portion of the reconfigurable fabric for loading other cores.

The general view of the scalable systolic array can be seen on Figure 1, where the specific region for allocating one or several scalable cores is shown.

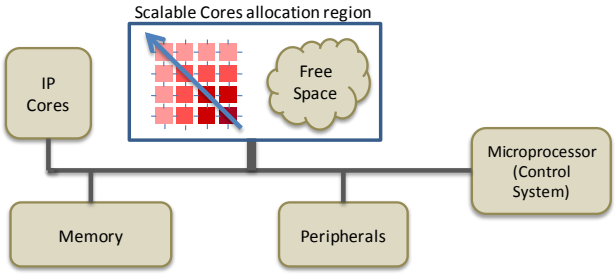

Figure 1. System general view. The core is scaled by means of the addition of processing elements

Similarly to the state of the art, the selected reconfiguration technique is partial dynamic reconfiguration. Due to this, the core scaling process does not disturb other cores that might be running in the system. Furthermore, the scalable core context switch is done at run-time, without stopping its functionality and, therefore pulling data out from the array is not required. In the following subsections, the proposed processing elements interconnections, as well as the design flow are shown.

3.1 Processing Elements interconnection

An important advantage of systolic arrays is the fact that signals entering or leaving a processing element are mainly addressed to its closer neighbors. As a result, the existing interconnections are very regular. In this work, north, south, east and west ports have been considered for each element.

According to the Modular Design flow [16], different modules have to be connected through bus-macros that allow signals to cross reconfigurable module boundaries. These macros are instantiated in the top design, and in order to act as hard modules that do not change, they are constantly reloaded in the systems with partial reconfigurations. Due to the nature of these static bus-macros and the related design flow, an important tradeoff between the core scaling granularity and the area losses appears. The FPGA resources required for the bus-macro implementation cannot be used by the core itself and thus it is desirable to keep its number as low as possible in order to reduce area overheads. Meanwhile, the less bus-macros are included in the systems, the bigger the reconfiguration granularity, and this results in restricted system flexibility. From the systolic arrays design point of view, the use of macro structures forces the selection of processing elements with higher granularity restricting the systems flexibility and resulting in systolic arrays architectures that are tightly coupled to a specific application. Furthermore, the integration of bus-macros restricts the use of the free area, from the one reserved for loading scalable cores, by other cores as cores to be loaded have to strictly fit into the area defined by two consecutive macros.

The problems of resource usage have been partially solved in the latest version of the EAPR flow [17], where single slice bus-macros have been introduced. Single slice bus-macros are part of the reconfigurable area, instead of being part of the static base design. As a result, fewer resources are consumed by the macros and also, resources are not used until a processing element is loaded in the reconfigurable area. However, this design flow does not permit to relocate a design along the defined reconfigurable area. Therefore, it is not suitable for the processing elements replication process, which is essential for achieving the flexibility and generality of the array proposed in this paper.

This paper provides a direct solution of the area/scaling granularity tradeoff by proposing a systolic array architecture that does not need bus-macros. This is achieved by exploiting the symmetry in the communications between processing elements in the systolic array. To achieve this, the north and west connections of an element have been designed using the same FPGA routing resources as the south and east connections. This symmetry has been also exploited to the design of the array processing elements that use the same routing resources to transmit the same signals. As a result, when a new element is added to the array in the reconfigurable area, its south routing wires will fit with the north wires of the element below, and their north wires with the south port of the element above. If the east-west connections keep the same conditions, this technique permits all the elements to be wire-compatible without the use of bus macros and the communications between the processing elements is guaranteed during dynamic reconfiguration. The symmetry of the north/south and east/west connections can be perceived from Figure 2, where three processing elements are included and the elements connections are highlighted.

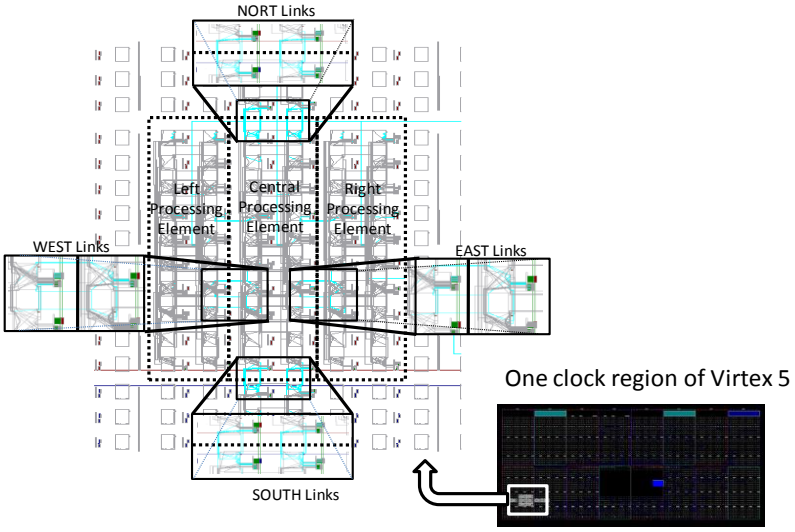

Figure 2. Symmetry of the north/south and east/west connections of the processing

Only when a block output has to be connected to several inputs of the adjacent element, an additional element, called anchor, is required. Anchor elements are implemented with look up tables and their main functionality is to distribute signals. The use of anchors permits to reduce the number of wires that cross the block boundaries. Anyway, anchor elements are considered part of each processing block.

In the use case and results section, some numerical results will be included to quantify the improvement of the non-bus macro approach.

3.2 Processing Elements Design Flow

In this subsection, a design flow to generate processing elements, customized to solve concrete problems and compatible with the generic systolic array approach is proposed. With this flow, specific scalable systolic solutions can be provided in a broad range of computational fields.

The first step of the flow is to define the systolic architecture that is required for the specific application. There exists extensive literature on this topic, including some automatic solutions [18]. Afterwards, the defined architecture is mapped into the reconfigurable system and divided in two parts: the systolic array itself and the control logic. The array is included in the reconfigurable region of the system and it is built and/or scaled with the method proposed in this paper. Differently, the control logic is included in the system static area, and is not affected by reconfigurations. Therefore, its design has to be valid for all the possible dimensions of the systolic array.

Once the architecture is mapped, the logic design of the basic elements of the array can be done. Each processing element is, indeed, a hard macro that can be directly designed with the FPGA Editor tool, or using a VHDL description that is tuned afterwards to define the shape of the element. After this, the processing element is instantiated five times in an ISE design with specific placement constraints, such that one of them is placed in the center and the other four blocks around it. Then, a step-by-step routing process is carried out in the FPGA Editor to define the connections between the central module and the adjacent ones. In this moment, the symmetry requirement of the communications, explained in the previous section, has to be accomplished. Finally, in order to generate the partial bitstream, the configuration frames that correspond to a single processing element are extracted from the full configuration with the five processing elements. This can be done using the bitgen Xilinx tool, or by means of a read-back operation of a processing element from the FPGA configuration memory.

By using this method, when the generated bitstream is replicated in the reconfigurable fabric, not only the processing elements logic content, but also, the communications among the blocks are automatically configured. This allows the creation and scaling of the systolic architecture by means of the replication, through relocation, of the unique single processing element. As a result, the total configuration data that has to be stored is that of a single processing element. The same relocation capacity is provided by the Modular Design flow, but with the expense of bus-macros. However, with the EAPR flow, a bitstream of the full systolic array has to be stored for each of the N possible scalability levels. This is because, on the contrary of the methodology proposed of this paper, the new macros that it uses do not permit to relocate modules and therefore module replication is not possible. The problem of the growth in the number of necessary bitstreams to manage the process of scaling with the EAPR flow can be seen in the experimental results of [14]. In the subsection 4.2, a quantitative estimation of these advantages will be provided.

Another important consequence of the relocation possibilities of the flow is that it permits to design processing elements independently from the system they will finally belong to, something that is not possible with the EAPR flow. This provides generality and permits: i) systolic cores to be configured from a library of processing elements or even, ii) to self-replicate a processing element that is already configured in the device and does not belong to the library.

The relocation of processing elements is performed by means of specific software functions, combined with the ICAP (Internal Configuration Access port) drivers provided by Xilinx. Furthermore, small bit manipulation techniques can be used to tune specific parameters of the processing element in order to build heterogeneous systolic architectures where each element has different parameter values, like filter constants or transform coefficients.

4 Results and Use Case

In this section, a general evaluation of the proposed architecture is shown, including a theoretical comparison with other existing reconfigurable methodologies to deal with scalable architectures. In addition, a use case is provided to check the validity of the design flow in order to adapt the generic architecture to a particular problem. Finally, some numerical results will be shown related with the implementation of the use case. The design has been implemented using a Virtex-5 FX70T FPGA from Xilinx with a PPC440 embedded processor.

4.1 General Evaluation of the System

In this subsection, the proposed system will be evaluated by comparing its advantages in terms of system memory requirements and reconfiguration time with common solutions that are based on the Modular Design flow and the EAPR. All the provided results are restricted to completely homogeneous squared systolic arrays, but conclusions can be extended to heterogeneous architectures.

In the proposed solution, the size of the bitstream that has to be stored for the unique processing element is:

$$BitstreamSize = Frames\_per\_CLB \times Column\_per\_element \times \lceil Rows\_per\_element \rceil \quad (1)$$

being *Frames_per_CLB* the number of configuration frames of each CLB column of the FPGA and $\lceil n \rceil$ the integer division of n, in this case the number of rows of the basic element (Rows_per_element in (1)), by the number of CLB Rows per configuration Frame of the device. Typical values for this parameter are 16 CLBs for each Virtex-4 configuration frame and 20 CLBs for Virtex-5. A direct consequence of the relocation possibilities explained at the end of subsection 3.2, the total configuration data that has to be stored is that of a single processing element. The same result can be achieved with the Modular Design flow, but without considering the bus-macros overhead, which is high as it will be shown further in this section.

On the contrary, regarding the EAPR design flow, a bitstream of the full systolic array has to be stored for each of the N possible scalability levels. The size of each bitstream can be calculated with:

$$BitstreamSize = Frames\_per\_CLB \times N \times Column\_per\_element \times \lceil Rows\_per\_element \times N \rceil \quad (2)$$

The total amount of configuration data that has to be stored in the system with EAPR design flow to allow the different levels of scalability is:

$$TotalConfigurationData = N \times BitstreamSize = N \times Frames\_per\_CLB \times N \times Column\_per\_element \times \lceil Rows\_per\_element \times N \rceil \quad (3)$$

Table 1 shows the improvements with respect to the total amount of necessary configuration data that derive from the proposed solution, comparing with the EAPR and the Modular Design flow. As a reference, improvements compared with a static, non scalable design of the maximum N×N size are included. As it can be seen, the exact value of this comparison depends on the relative size of the processing element respect to the size of the reconfiguration frame. However, the maximum bound is provided to give a general idea of the achieved advantages.

Table 1. Comparison of the total amount of configuration bits of each technique

Flow	Relation with the proposed flow	Maximum Bound
Static Design	$\frac{N \times \lceil Rows\_per\_element \times N \rceil}{\lceil Rows\_per\_element \rceil}$	N^2
Modular Design	1	1
EAPR	$\frac{N \times N \times \lceil Rows\_per\_element \times N \rceil}{\lceil Rows\_per\_element \rceil}$	N^3

Regarding the time overhead for the scaling process, the use of the EAPR and the modular design flows will be compared with the proposal of this paper. In the present approach, the necessary reconfiguration time to scale the systolic array from (N-1) × (N-1) to N×N dimensions is the time to reconfigure the 2N-1 new elements. That is:

$$ReconfigurationTime = (2N-1) \times ReconfigurationTime\_per\_CLB \times Column\_per\_element \times \lceil Rows\_per\_element \rceil \quad (4)$$

The same result can be obtained with the modular design flow, again not considering the area overflow of bus-macros. However, with the EAPR, it is necessary to reconfigure the full core ($N_{max} \times N_{max}$ dimensions) in order to perform the process of scaling. The time overhead is:

$$ReconfigurationTime = ReconfigurationTime\_per\_CLB \times N_{max} \times Column\_per\_element \times \lceil Rows\_per\_element \times N_{max} \rceil \quad (5)$$

This parameter has no meaning in the case of the non scalable systolic design. The comparison with the proposed method results is shown in Table 2.

Table2. Comparison of the reconfiguration time overheads respect to the proposed method.

Flow	Relation with the proposed flow	Maximum Bound
Modular Design	1	1
EAPR	$\frac{N_{max} \times \lceil Rows\_per\_element \times N_{max} \rceil}{(2 \times N - 1) \times \lceil Rows\_per\_element \rceil}$	$\frac{N \times N}{(2 \times N - 1)}$

4.2 Use Case Design

An image filter that performs window–based operations using a reconfigurable mask has been developed as an example of the proposed architecture as well as the design flow. This operator is the base of several image processing applications. Its principle is the application of an N×N pixels window to the image, operating the selected pixels according to a mask, and obtaining an output result. Usually, the output of the operation is the result in the position of the central point of the window. This window is slid across the whole image, generating all the points of the processed image. In [7], a very good review on this kind of operators is provided.

The operation developed in this paper is the 2D convolution, which is a special case of these windows-based operations. The output is a weighted average of the input pixels inside the window, using the mask like the weights. With the technique proposed in this paper, it is possible to create a scalable two-dimensional reconfigurable convolution, with the property of modifying its weights and its size in real time.

The systolic structure developed for the filter includes a static region with some control logic and memory elements to provide data in the correct order, and the systolic array itself, which is the scalable element. Following the provided design flow, the basic processing element of the systolic region of the filter has been designed using the FPGA Editor tool. Afterwards, the symmetric connections have been created and a partial bitstream for the element has been generated by reading back the corresponding portion of the FPGA.

Finally, to communicate the core with the static region and to bypass the columns of the FPGA whose content are not CLBs, as a first approach, static bus-macros have been used. Designed bus-macros have to fulfill also with the symmetry requirement for the communications among the processing elements. This guarantees that the bordering processing elements fit with the static bus-macros, in the same way they fit with the other processing elements. The main drawback of this approach is that it renders DSPs and BRAMs columns. In addition, the size of each core is limited to the number of columns between each two non CLB columns. However this problem, that is common in runtime reconfigurable system, is minor in the proposed in this paper architecture, since it focuses on small grain processing elements. The system has been tested in terms of the scaling process by loading subsequently several processing elements in the systems using partial reconfiguration. As a result the systolic architecture and the no-macro based approach have been successfully validated.

Finally, in the Figure 3, layout captures of the FPGA Editor are provided, in order to show the result of the process of scaling the systolic architecture from dimensions 3 × 3 to 5 × 5. The selected FPGA layout allows a maximum size of 7 × 7 elements.

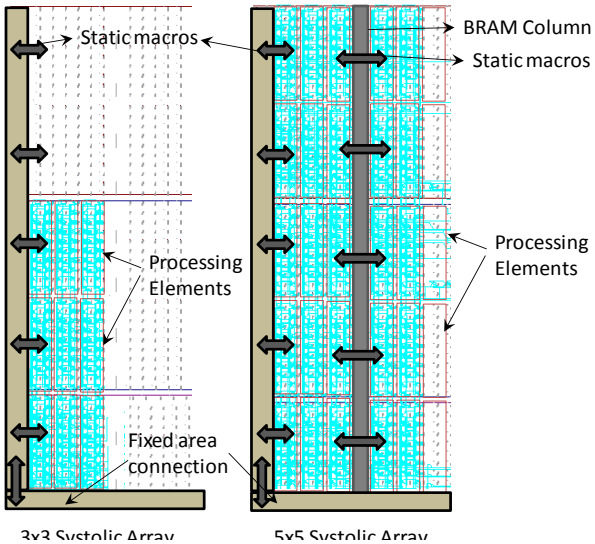

Figure 3. FPGA Editor layout of the core scaled from dimensions 3×3 to 5×5 whith highlighted static macros

4.3 Use Case Results

While in the previous sections, some quantitative advantages of the proposed architecture have been underlined, in this subsection, some implementation results obtained from the use case will be provided, in order to prove the mentioned advantages in this particular design.

The first comparison is done in terms of logic consumption, comparing the proposed connections among elements without bus-macros, with respect to an implementation using Xilinx bus-macros. As it has been already mentioned, the basic processing element has four connections: North and south connections are 10 bits wide, while east and west connections are 16 bits wide. Since Xilinx dual-slice bus-macro allows an 8-bit communication between two reconfigurable regions, two bus-macros per interface would be required to allow all necessary interconnections. Consequently, the overhead of including these bus-macros are 8 CLBs for each processing element. Since each basic element occupies 20 CLBs, the area consumption of the elements with bus-macros would increase a 40%, compared with the solution proposed in this paper.

Additionally, it can be shown that the total amount of configuration bits is also reduced. As it is shown in Figure 3, each processing element occupies 2 columns and 10 rows in the reconfigurable device. Since each Virtex-5 CLB column requires 36 reconfiguration frames, 2×36 full frames have to be stored for the basic processing element. Differently, in a static and non-scalable solution, a bitstream of the area that corresponds to the biggest possible block should be stored. Regarding this use case, the 7 × 7 core occupies 14 columns of 70 CLBs each one. To configure each column, data corresponding to 4 clock regions are necessary. Since each column has 36 frames, 2016 frames have to be stored to configure the core. Moreover, in the EAPR design flow, considering the 7 scalability possibilities, the storage necessity is 7×2016 frames. A

summary of this comparison is shown in Table 3. The final measured memory occupation of the basic processing element is 11 Kbytes.

Table3. Amount of configuration frames with each technique for N_{max} = 7.

Flow	Number of necessary frames	Relation with the proposed flow
Proposed	72	1
Static Design	2016	28
Modular Design	72	1
EAPR	14112	196

Finally, regarding the reconfiguration time, the EAPR design flow always consumes the necessary time to reconfigure the full 7 × 7 architecture, while both the proposed and the modular flows, only have to reconfigure the new elements. The FPGA Frames that have to be reconfigured to scale the systolic array from (N-1) × (N-1) to N×N dimensions can be seen in Table 4. A comparison between the two options is also shown, for different N values. The number of frames to reconfigure, in the case of the design flow proposed in this paper, depends on the value of N, but in the worst case, it is 2.15 times better than the achieved with the EAPR flow. The measured time to reconfigure each processing element is 0.34 ms.

Table4. Comparison of the reconfiguration time of the EAPR flow respect to this proposal.

N	Number of frames to reconfigure with the proposed flow	Number of frames to reconfigure with EAPR	Relation
1	72	2016	28
3	360	2016	5.6
5	648	2016	3.11
7	936	2016	2.15

5 Conclusions and Future Work

This paper describes the architecture of a generic systolic array with spatial scalability capability. The method is based on the replication and relocation of the single element using dynamic partial reconfiguration a basic element to generate an array which size could be adapted at runtime to the available area in the device, or to the application requirements of the executed task. The released area in the device can be freely used to load other cores. To allow the process of scaling, a communication structure that does not require the use of bus-macros is proposed, resulting in important area savings and improved FPGA occupation. Scalability of the solution is guaranteed in non homogeneous FPGAs (with embedded RAMs and other predefined blocks) by a symmetric bus-macro based feed-through structure, compatible with the scalable part of the architecture. In addition, a proprietary design flow is provided to adapt the generic architecture to the solution of specific problems. This allows flexibility enhancements with respect to the state of the art alternatives. In addition, with this approach, improvements are also achieved for both, the reconfiguration time overhead and the amount of configuration data. An image filter has been developed as a use case example.

Future work will include the development of a library of basic processing elements to provide scalable solutions in different data treatment fields, as well as to automate

the decision of run-time scaling the core, according to changing application requirements or system conditions.

6 Acknowledgements

This work was supported by the Spanish Ministry of Science and Research under the project DR.SIMON (Dynamic Reconfigurability for Scalability in Multimedia Oriented Networks) with number TEC2008-06486-C02-01.

References

1. Xun ZHANG, Hassan RABAH, Serge WEBER, "Auto-adaptive reconfigurable architecture for scalable multimedia applications," In the Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems, pp. 139-145, 2007
2. Syed, S.B.; Bayoumi, M.A., "A scalable architecture for discrete wavelet transform," In the Proceedings of the Conference on Computer Architectures for Machine Perception, 1995. CAMP '95, vol., no., pp.44-50, 18-20 Sep 1995
3. Jian Huang; Jooheung Lee; Yimin Ge, "An array-based scalable architecture for DCT computations in video coding," In the Proceedings of the International Conference on Neural Networks and Signal Processing, 2008, vol., no., pp.451-455, 7-11 June 2008
4. Meher, P.K.; Chandrasekaran, S.; Amira, A., "FPGA Realization of FIR Filters by Efficient and Flexible Systolization Using Distributed Arithmetic," In IEEE Transactions on Signal Processing, vol.56, no.7, pp.3009-3017, July 2008
5. Yin-Tsung Hwang; Ying-Ji Chen; Wei-Da Chen, "Scalable FFT kernel designs for MIMO OFDM based communication systems," In the TENCON 2007 - 2007 IEEE Region 10 Conference ,vol., no., pp.1-4, Oct. 30 2007-Nov. 2 2007
6. Krishnaiah, G.; Engin, N.; Sawitzki, S., "Scalable Reconfigurable Channel Decoder Architecture for Future Wireless Handsets," In the Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07, vol., no., pp.1-6, 16-20 April 2007
7. Saldana, G.; Arias-Estrada, M., "FPGA-based customizable systolic architecture for image processing applications," In the Proceedings of the International Conference on Reconfigurable Computing and FPGAs, 2005. ReConFig 2005., vol., no., pp.8 pp.-3
8. Huang, Jian; Lee, Jooheung, "A Self-Reconfigurable Platform for Scalable DCT Computation Using Compressed Partial Bitstreams and BlockRAM Prefetching," In the Proceedings of the IEEE Computer Society Annual Symposium on VLSI, 2009. ISVLSI '09., pages.67-72, May 2009
9. Mehendale, M.; Sherlekar, S.D.; Venkatesh, G., "Area-delay tradeoff in distributed arithmetic based implementation of FIR filters," In the Proceedings of the Tenth International Conference on VLSI Design, 1997, pages.124-129, Jan 1997
10. Danne, K., "Distributed arithmetic FPGA design with online scalable size and performance," In the Proceedings of the 17th Symposium on Integrated Circuits and Systems Design, 2004. SBCCI 2004, pp. 135-140, Sept. 2004
11. Selepis, I.N; Bekakos, M.P., "VHDL Code Automatic Generator for Systolic Arrays," In the Information and Communication Technologies, 2006.(ICTTA '06). vol.2, pp.2330-2334
12. Zhao Junchao; Chen Weiliang; Wei Shaojun, "Parameterized IP core design", In the Proceedings of the 4th International Conference on ASIC, 2001., pages.744-747, 2001
13. Yeong-Jae Oh; Hanho Lee; Chong-Ho Lee, "A reconfigurable FIR filter design using dynamic partial reconfiguration," In the Proceedings of the IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006. 2006, pages.4 pp.-4854
14. J. Huang, M. Parris, J. Lee, and R. F. DeMara, "Scalable FPGA Architecture for DCT Computation using Dynamic Partial Reconfiguration", In the Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), Las Vegas, U.S.A., July 2008.
15. Gause, J.; Cheung, P.Y.K.; Luk, W., "Reconfigurable shape-adaptive template matching architectures," In the Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2002. vol., no., pp. 98-107, 2002
16. Xilinx Corp., "XAPP 290: Two flows for Partial Reconfiguration: Module Based or Difference Based," www.xilinx.com, Sept 2004.
17. Xilinx Inc. 2006. Early Access Partial Reconfiguration User Guide. Xilinx Inc., San Jose.
18. Ko, C.K.; Wing, O., "Mapping strategy for automatic design of systolic arrays," In the Proceedings of the International Conference on Systolic Arrays, 1988, vol., no., pp.285-294, 25-27 May 1988